PRÉPROCESSEUR DE MACROS COMMANDES

PROJET DU MODULE «INITIATION À LA PROGRAMMATION»
LICENCE INFORMATIQUE 1999/2000

*Le but du projet est la construction d'un préprocesseur de macros-commandes analogue (*en partie*) à la commande* `cpp` *(équivalente à* `gcc -E`*), le préprocesseur du langage C.* On souhaite atteindre ce but en produisant un ensemble de modules lisibles, maintenables, portables, extensibles et réutilisables.

1 La commande cpp

Sous emacs, la commande `c-h i` vous permet d'accéder à la documentation de la commande `cpp`. Vous pouvez aussi vous reporter au Chapitre 5 du livre écrit par Jean-Pierre Braquelaire – Méthodologie de la programmation en C, MASSON. Vous trouverez un fichier test sous :
`http://dept-info.labri.u-bordeaux.fr/~wacren/IP/macros.c`

2 Les fonctionnalités demandées

Le but du projet n'est pas de programmer toutes les fonctionnalités de la commande `cpp`, mais d'écrire une application (un ensemble de modules) extensible à celles-ci. Nous avons regroupé les fonctionnalités concernées par notre projet en deux catégories :

1. les fonctionnalités de base (celles que votre application doit prendre en charge),
2. les fonctionnalités avancées (les extensions que l'on pourra facilement intégrer à une nouvelle version).

2.1 les fonctionnalités de base

1. **Lecture / Ecriture** : votre commande doit être capable de reproduire (sur la sortie standard) un texte lu (sur l'entrée standard) et se terminant par EOF (c-d); pour ce faire, on utilisera des fonctions des familles `int getchar(void)` et `putchar(int c)` (voir le «man» de ces fonctions).
2. **Macros-constantes** : prise en compte des directives `define` (*sans paramètre*) et `undef`. En particulier, on prendra en charge les définitions en cascade des macros (`cpp` s'applique récursivement à la définition d'une macro sauf si celle-ci est auto-référentielle, voir le chapître «Macro Pitfalls» du manuel sous emacs).
3. **Inclusions conditionnelles** : prise en compte des directives `ifdef`, `ifndef`, `else` et `endif`. On tiendra compte de la possible imbrication de ces directives.
4. **Macros prédéfinies** : `__LINE__`, `__TIME__` ou encore `__DATE__`, pour ces deux dernières on utilisera la bibliothèque `time.h`

2.2 fonctionnalités avancées

1. Gestion des paramètres de la fonction :
   `cpp [-Dname[=definition] [infile|-] [outfile|-]`
   avec
   - `-D NAME` attribue la définition 1 à la macro `NAME`
   - `-D NAME=DEFINITION` attribue la définition `DEFINITION` à la macro `NAME`
   - `infile|-` définit le fichier en entrée (- signifiant entrée standard)
   - `outfile|-` définit le fichier en sortie (- signifiant stortie standard)
2. Inclusion de fichiers : prise en compte de la directive `include`.
3. Macros-fonctions : prise en compte des macros paramétrées.

3 Quelques conseils

Dans un premier temps il s'agit de structurer votre application (on parle d'architecture du logiciel) par exemple :

1. entrées/sorties de bas niveau ;
2. analyse lexicale du texte source :
   - identification des séparateurs (espace / – {}... `\n` `\t`),
   - identification des mots ;
3. analyse sémantique du texte source : interprétation des mots suivant un contexte, ce peut-être :
   - une chaine de caractères,
   - un commentaire,
   - une directive,
   - un paramètre de directive (nom de macro, une définition de macro, nom de fichier...)
   - un mot ordinaire.
4. exécution des directives.

Ensuite, il s'agit de dégager un ensemble d'outils (ex. traitement de chaines de caractères, stockage d'informations) et de coder chaque outil en respectant le principe du masquage de l'implémentation.

On construira alors une première version du logiciel que l'on enrichira progressivement.

On visera la réutilisabilité et l'extensibilité, il sera nécessaire de réécrire plusieurs fois le même module : le but de l'exercice est d'obtenir un code d'excellente qualité.

N'oubliez pas de respecter les «copyrights» des textes sources que vous pourriez avoir repris voire modifiés.

4 Lectures conseillées

- J.P. Braquelaire - Méthodologie de la programmation en C, Chapître 8.
- R. Strandh - Imperative programming using C - (Unité de Maîtrise «Techniques et Fondement de la programmation»). `http://dept-info.labri.u-bordeaux.fr/~strandh/`
- R. Strandh - Modular C `http://dept-info.labri.u-bordeaux.fr/~strandh/Teaching/reuse.ps`

5 Évaluation

Le projet ne sera pas directement noté : l'évaluation (en temps limité et sur papier) portera sur la modification ou l'ajout de fonctionnalités.